TILING SYSTEM FOR 3D RENDERED GRAPHICS

Field of the Invention

This invention relates to the generation of 3D computer graphics images and describes improvements to techniques described in our British patent 2343603, and earlier related patent, GB2298111.

Background of the Invention

In these systems, the 2D image plane is subdivided into preferably rectangular regions of pixels and, for each such region, a list of the objects that are 'of interest' to that region is compiled. These regions will be referred to as tiles. When the list for a particular tile contains all such objects, the tile can be rendered entirely. An alternative scheme, (as described in our patent application PCT/GB01/02536), allows for partial rendering whereby the list may simply contain a subset of the required objects.

It is important to have cost-effective means of determining which primitives, typically triangles and line segments, should be placed in which tiles. An example is given in Figure 1. This shows the virtual image plane (e.g. framebuffer) '1', consisting of a collection of pixels, '2'. These pixels are grouped by tiles, '3'. An example primitive '4', a triangle, is shown. This would then need to be placed or referenced by at least the lists for the shaded tiles, 5. Note that it is permissible to include the primitive in additional tile lists, e.g. such as for tile '6', but it would be incorrect to leave it out any of the shaded tiles.

The process of determining which primitives belong in which tiles is called 'binning' or 'tiling'.

Modern 3D computer games tend to have very high polygon counts and thus a high number of primitives. During the generation of the scene, many of the primitives become too small to be sampled and thus do not affect the image. This happens as the objects which they are used to represent recede

into the distance. Storing these very small primitives in tile lists is an unnecessary use of time, memory, and processing effort. In Figure 7, a simple example of a small triangle primitive, '53', is shown against a grid of image pixels, '50'. Typically, pixels, for example '51', are sampled at one point, say the top left corner, '52'. The triangle, '53' fails to cross any of the sample locations and so will not affect the image. In this particular example, this fact can be trivially determined by considering the primitive's bounding box, '54'. Because this does not contain any 'integer' sampling locations, the primitive is redundant.

In Figures 8 and 9, other small triangles that miss all the sampling points are shown. In these cases, however, a bounding box rejection test will fail to cull these primitives. One naïve cost-saving approach might be to compute the area of the primitive and, if it is small, reject the surface. Unfortunately, this does not always work as an object may have been modelled by, say, by tessellation to small triangles- rejecting all small triangles could easily make the entire object disappear!

In 3D computer graphics, it is common for triangular/convex polygonal primitives to be defined with an associated 'winding order' which is typically one of {Both, Clockwise, Anticlockwise}. The winding order is used as a performance optimization to eliminate some primitives prior to rendering. This optimization functions such that if, after projection, the primitive has an order of vertices that does not match its specified winding order, that primitive can be ignored.

In the IEEE 754 standard, floating point numbers are represented using a sign bit value, sign, an exponent value, ex, and an 'M' bit mantissa value, m, which with the exception of some minor special cases which can be ignored, represents a value equal to $(-1)^{sign}.2^{ex}.\left(1+\frac{m}{2^M}\right)$, where $0 \le m < 2^M$. For 32-bit IEEE floating point numbers, M is 23, but reduced precision maths could use fewer bits. If a system were converting a normalized

IEEE float, x, to a system that uses M bits for the mantissa, then it can be shown that the error in the approximated x value, $\underset{M}{x}$, obeys the inequality...

$$err\left(\underset{M}{x}\right) \leq \frac{1}{2^{M+1}}|x| \quad \text{..... eqn 1}$$

Similar expressions for addition and multiplication operations can be derived:

$$err\left(\underset{M}{x}+\underset{M}{y}\right) \leq err\left(\underset{M}{x}\right)+err\left(\underset{M}{y}\right) \quad \text{..... eqn 2}$$

$$err\left(\underset{M}{x}.\underset{M}{y}\right) \leq err\left(\underset{M}{x}\right)|y|+err\left(\underset{M}{y}\right)|x|+err\left(\underset{M}{x}\right)err\left(\underset{M}{y}\right) \quad \text{..... eqn 3}$$

In GB2343603, a method of determining which primitives (i.e. the basic rendering components used to represent objects) are in a particular tile is presented. For each primitive (in particular, a triangle) it first determines a bounding box of candidate tiles and then attempts to eliminate a candidate tile by testing each of the edge equations against certain corner points of that tile. For each edge, only one corner of each tile needs to be tested. If the candidate is not eliminated, then the primitive is added to the tile's list.

There are several inconvenient aspects with the approach described in that patent:

(1) The precision of the mathematics used to perform the rejection calculations must be at least as accurate as, or at least give identical results to, that used by the rendering system when that decides which pixels a primitive covers.

An example of a potential problem is shown in Figure 2. A 2x2 subset of the image's tiles, '7', consisting of a number of pixels, 8, is shown along with a boundary edge of a primitive, '9'. Those pixels, shown in grey, represent the pixels that may be deemed to be 'inside' the primitive by the

renderer. Obviously, any tile that contains such a pixel, even if there is only one such pixel, e.g. '10', should also have the primitive included in its list.

This may seem like a triviality, but in practice it may be more complicated. For example, floating point arithmetic may be used to perform the various inside/outside tests, and when rendering a particular tile, the coordinate system may be re-aligned so that rather than using a pixel origin that is at the top left of the image, the rendering may use a local origin, say, at the top left of the tile. Doing this reduces the cost of the hardware needed when rendering inside the tile, however, it would make it very difficult and/or more expensive to have a 'global test' with exactly the same accuracy that can be used for the 'tiling' operation. If there is any deviation in accuracy, then there is a chance that tile '11' might be deemed to not include the primitive and hence cause pixel '10' to be rendered incorrectly.

Even if the mathematics provides exactly the same results as that used inside the renderer, a hardware solution would require a significant amount of silicon.

An alternative could be to use fixed-point mathematics, rather than floating-point, but this incurs the problem that arbitrary polygons would have to be 'clipped' to the valid supported mathematical range. Clipping is an expensive operation which is preferable to avoid.

(2) The 'test corner' of the tile varies depending on the parameters of the particular edge's line/plane equation. This is an inconvenience, as it requires decisions on a tile-by-tile/edge-by-edge basis.

(3) The system only supported triangles and it is useful to also apply the 'tiling' process to other primitive types such as 'thick lines' constructed from parallelograms. Of course, it is trivial to either convert a parallelogram to a pair of triangles, or even extend the method to support any convex polygon, but for the case of lines a more efficient approach is desirable.

(4) The system does not attempt to address cases where 'tiny' primitives, e.g. those that are around the size of a pixel, are supplied to the tiling engine but don't actually get rendered. Many such primitives, such as those illustrated in Figures 6 thru 8 simply will not affect the outcome of the final image because they fall between the 'sampling points' used in the renderer. Adding these to the tile list is thus wasted effort.

Summary of the Invention

Preferred embodiments of the invention presents a method of handling not only the simple case shown in Figure 6, but also those that arise in Figures 7 and 8.

We have appreciated that although it is desirable to add each (convex polygon) primitive to the minimum set of tile lists, it may be expensive to compute this minimal set and that it is certainly safe to occasionally add extra unneeded tiles, such as '6' shown in Figure 1. It thus becomes apparent that lower precision maths, i.e. cheaper to implement, could be used provided that, if it erred, it always did so in favour of including additional tiles. More precisely, the set of tiles computed by a lower precision approach must be a superset of the correct set of tiles.

As a first embodiment of the invention, a method and an apparatus are presented which safely use lower precision calculations. Furthermore, the system may be tuned to trade-off accuracy, i.e. implementation cost, of the maths against the increased likelihood of additional redundant tiles. The system also has the advantage of using a uniform testing procedure for all edges/tiles.

In a second embodiment of the invention, we have also appreciated that a method of directly tiling line segment primitives that takes advantage of special properties, is desirable. The method presented is cheaper than converting the primitive to an equivalent polygon or polygons.

In a third embodiment, the invention provides a means of rejecting certain 'small' primitives that are guaranteed not to influence the image.

Brief Description of the Drawings

Preferred embodiments of the invention will now be described in detail by way of examples with reference to the accompanying drawings in which:

Figure 1 shows a triangular primitive projected onto an image plane and illustrates the minimal set of tiles which are affected by the primitive;

Figure 2 shows a 'close up' of a region of four tiles and a primitive edge in which one tile contains only a single pixel that is affected by the primitive;

Figure 3 shows an overview of the steps performed when determining if and how to 'tile' a primitive using the invention;

Figure 4 shows the steps performed for the tiling of a primitive that has not been rejected due to tiling;

Figure 5 shows a tile partially covered by a primitive with a 'safe' test point for the edge, as described in the prior art of GB2343603, and the preferred sampling location for an embodiment of the invention;

Figure 6 shows the shape and orientation of preferred line segment primitives;

Figure 7 shows an example of a small primitive that will not be visible if passed to the renderer, but could be rejected or culled using a simple bounding box test;

Figures 8 and 9 show examples of small primitives that also will not be visible, but cannot be culled simply using bounding boxes; and

Figure 10 shows the steps used in testing whether small polygons can be culled.

Detailed Description of Preferred Embodiments

A preferred embodiment of the invention will be presented with reference to Figure 3. As a first step, a primitive, which is preferably a triangle, is input, '14'.

The triangle has vertices, $\{(x_0,y_0),(x_1,y_1),(x_2,y_2)\}$, which are specified using IEEE, 32-bit, floating-point coordinates. As with GB2343603, the bounding box of the primitive is identified by computing the maximums and the minimums of both the projected X and Y coordinates values, '15', i.e.

$$p_{x\min} = \min(x_0,x_1,x_2)$$
$$p_{x\max} = \max(x_0,x_1,x_2)$$
$$p_{y\min} = \min(y_0,y_1,y_2)$$
$$p_{y\max} = \max(y_0,y_1,y_2)$$

These maximums and minimums are determined at full accuracy as this is a relatively cheap operation. As an optimization, it is also possible to test for equality of the vertex positions and reject those triangles that have repeated vertex positions, as these have exactly zero area and will not contribute to the image. At this point, the invention can also reject any primitives that are trivially completely 'off-screen'.

Using the computed bounding box and primitive's coordinates, small primitive culling tests are performed, '16', details of which are given later. If the primitive is not deemed to affect the image, it is discarded, '17', otherwise it is 'tiled', i.e. placed into the relevant tile lists, '18'. The details of the tiling process for the triangle primitive, '18', will now be presented along with extensions to process wide lines. The small object culling, '16' is presented after that.

Our British patent No. 2343603 describes how the edges of a (convex polygonal) primitive can be represented as lines (or plane equations) of the form...

$$Ax + By + C = 0$$

…where A, B, and C are coefficients that depend on the position and orientation of the particular edge and x and y are pixel coordinates. A point, (x,y), in the image plane is defined as being strictly inside a 'clockwise' primitive when…

$$A_i x + B_i y + C_i > 0$$

…for all edges, i. Under certain circumstances (e.g. the fill rules described in the computer graphics standard, OpenGL), some edges are also included as being inside the primitive. For the purposes of tiling, where the system can afford to be conservative, all edges can be considered to be part of the primitive and so the inside test for a 'clockwise' primitive is relaxed to:

$$\forall i, \quad A_i x + B_i y + C_i \geq 0$$

'Anticlockwise' primitives should use a '≤' test. For brevity, the document will only describe the clockwise case, as the extension to the other cases will be obvious to one skilled in the art.

For a projected triangle, GB 2343603 also gives the equations for computing the coefficients for the edge test, i.e., if (x_i, y_i) and (x_{i+1}, y_{i+1}) are the end points of an edge, i, then

$$A_i = y_i - y_{i+1}$$
$$B_i = x_{i+1} - x_i$$
$$C_i = x_i y_{i+1} - x_{i+1} y_i$$

These equations are actually derived from the adjoint of a 3x3 matrix that contains just the projected coordinates of the triangle vertices. In some situations it is preferable to calculate the (projected) edges of the triangle without having to project the vertices, i.e. using the homogeneous

coordinates. In that situation, the edge equations can also be computed using the corresponding adjoint matrix and, in an alternative embodiment, the tiling operation can be performed using homogeneous coordinates. For brevity, that alternative embodiment will not be presented, but the adaptation will be clear to one skilled in the art.

The overview of the processing steps contained in '18' in the preferred embodiment is now presented with reference to Figure 4.

An initial rectangular region of candidate tiles, limited to the extent of the image plane, is then determined from the extremities, '22' of the primitive.

As an optimization in the preferred embodiment, when the set of candidate tiles is only a single tile high and/or a single tile wide, '23', the bounding box can be used directly as all tiles will be required, '24'.

The next step is to compute the parameters, '25' of the edge equations. This processing step differs from that of the earlier systems and will be described in more detail shortly.

The process continues by iterating through the candidate set of tiles, '26', testing each candidate against each of the edges, '27'. Only if all edge tests 'pass', the primitive is included in that tile's list. The edge test employed in the invention is also simpler than that presented in GB2343603. In an alternative embodiment, the system could use an alternative order to process the candidate tiles, such as a binary chop, which would eliminate some tiles based on the results of their neighbors.

To understand the edge equation calculations performed in '25', references will now be made to Figure 5. For a given candidate tile, '30', we must consider each of the edges of the primitive. In this example, an edge, '31' crosses the tile. The shaded region, '32', represents part of the area inside the primitive. As discussed in GB2343603, although performing an inside test at corner '33' for this particular edge will produce the correct result for this particular edge,

it is inconvenient to have to select the sample point on an edge-by-edge basis. Instead, it is preferable to use a uniform sampling location. In the preferred embodiment, this is chosen to be the top left corner '34'. The actual edge equations are adjusted slightly to allow this to be done.

Each edge is computed independently and this is now described. We define the end points (in order of definition) of edge 'i', in original full floating-point precision, to be (x_i^f, y_i^f) and (x_j^f, y_j^f) where $0 \le i \le 2$ and $j = (i+1) \bmod 3$. Two comparisons of these data values, $x_j^f \ge x_i^f$ and $y_j^f \ge y_i^f$, are performed and are used as described in the following pseudo code. This pseudo code can be implemented by those skilled in the art in either hardware or software.

```
OrientationTypes = {TopLeft, TopRight, BottomLeft,
BottomRight};

OrientationTypes Mode;

BOOL XjGTE, YjGTE;

// Compare the coordinate values
XjGTE := (xj >= xi);
yjGTE := (yj >= yi);

// Determine edge orientation…
IF (XjGTE AND NOT YjGTE)
{
     Mode := TopLeft;
}
ELSE IF (XjGTE AND YjGTE)
{
     Mode := TopRight;
}
ELSE IF (NOT XjGTE AND YjGTE)
{
     Mode := BottomRight;
}
ELSE
{
     Mode := BottomLeft;
}
```

The orientation of the edge determines the adjustments made to the edge equations. This involves computing 'shift' amounts as described in the following pseudo-code; the T_x and T_y values represent the dimensions of a tile in the x and the y directions respectively.

```
IF ((Mode == ToTopLeft) OR (Mode == ToBottomLeft))
{
     x_shift = Tx;
}
ELSE
{
     x_shift = 0;
}
IF ((Mode == ToTopLeft) OR (Mode == ToTopRight))
{
     y_shift = Ty;
}
ELSE
{
     y_shift = 0;
}
```

The logic behind this shift amount is that, rather than moving test point as a function of the edge, each edge will be shifted so that the same test corner can be used for all the edges. It will shortly become apparent to one skilled in the art that it is preferable to choose tile dimensions which are exact powers of two in size, for example, 32x16, as this will result in lower cost implementations.

This shift value is used to compute a new C_i coefficient for the edge based on the equation $C_i' = x_1 y_2 - x_2 y_1 + x_{shift}.A_i + y_{shift}.B_i$. This equation, however, cannot be used directly unless full precision mathematics is employed. Instead we use a 'safe' calculation that can be done with reduced precision.

Floating-point mathematics is used with a small number of mantissa bits, M. This will typically be between 8 and 22 bits, with the preferred embodiment using 16 bits.

From inequalities 1, 2, and 3, above, it is apparent that the errors in the floating-point numbers are bounded by their magnitudes. Using these magnitudes directly is inconvenient and so the embodiment simplifies the tests by relaxing the

error bounds further. Therefore, as a first step to computing a 'safe' edge equation, the following values, $xmag_i$ and $ymag_i$, are computed from the end points such that the following inequalities are obeyed:

$$2^{xmag} \geq \max(|x_j|,|x_i|,T_x) > 2^{xmag-1}$$
$$2^{ymag} \geq \max(|y_j|,|y_i|,T_y) > 2^{ymag-1}$$

It should be apparent to one skilled in the art, that this is a simple operation when applied to floating point numbers.

From these values and the application of the equations 1, 2, and 3, a 'safety margin' value, fslop, is computed thus:

$$fslop_i = 13 * 2^{xmag+ymag-(M+1)}$$

Again, this is a simple operation in the floating point involving a 'constant' mantissa value (derived from the '13') and some simple integer mathematics to derive the floating-point exponent.

Using the reduced precision mathematics, the edge equation coefficients are calculated as follows:

$$\underset{M}{a_i} = \underset{M}{y_i} - \underset{M}{y_j}$$
$$\underset{M}{b_i} = \underset{M}{x_j} - \underset{M}{x_i}$$
$$\underset{M}{c_i} = \underset{M}{x_i}\,\underset{M}{y_j} - \underset{M}{x_j}\,\underset{M}{y_i} + x_{shift}.\underset{M}{a_i} + y_{shift}.\underset{M}{b_i} + fslop_i$$

(Note, for anticlockwise primitives, the fslop value should be subtracted in the last equation).

Once these values have been computed for each edge, the system can iterate through the set of candidate tiles, '26'.

The inequality test for each edge in each tile, performed in '27', can be done in several ways. One such possibility is...

$$a_{i_M} X + b_{i_M} Y + c_{i_M} \geq 0$$

...where X and Y are the coordinates of the top left corner of the candidate tile, but the preferred embodiment uses the following...

$$b_{i_M} Y + c_{i_M} \geq -a_{i_M} X$$

...a floating-point magnitude comparison is slightly less expensive than an addition and (2) since this formulation also allows the left hand side to be held constant while iterating through a row of candidate tiles.

Although floating-point arithmetic has been used in the preferred embodiment, it would also be feasible with some adjustments to employ integer calculations.

Although unit '26' has been described as 'stepping' or 'iterating' through the candidate tiles, in an alternative embodiment, several candidate tiles could be tested simultaneously.

In an extension to the preferred embodiment, support for wide line segments is also provided, and would re-use many of the facilities provided for the tiling of other triangle/convex primitives. These line segments are preferably of the form shown in Figure 6, and fall into two classes - those that are predominantly vertical, such as that shown by '40', and those that are predominantly horizontal, such as '41'. They are specified by defining the end points of the central axis, i.e., '42' (x_0, y_0), and '43', (x_1, y_1). A line segment is chosen to be of the 'mainly vertical' form if $|\Delta_x| < |\Delta_y|$ and 'mainly horizontal' otherwise. The 'vertical' lines segments are capped by a horizontal line of a known width wx, '44', while the 'horizontal' lines are closed by a vertical section of an alternative known width wy, '45'.

As with the triangle primitives, the first step is to compute the bounding box of the candidate tiles. Because the

preferred embodiment avoids the use of full accuracy mathematics for cost saving purposes, it is not possible to know with 100% certainty which class a particular line segment falls into. For this reason the bounding box calculation will assume that the line segment could have either horizontal or vertical ends. As before, the system can compute min and max values of the end points cheaply. For the preferred embodiment, a bounding box on the segment is set to be:

$$(x_{bound\,\min}, x_{bound\,\max}) = \left(\left(\min(x_0, x_1) - \frac{w_X}{2}\right), \left(\max(x_0, x_1) + \frac{w_X}{2}\right)\right)$$

$$(y_{bound\,\min}, y_{bound\,\max}) = \left(\left(\min(y_0, y_1) - \frac{w_Y}{2}\right), \left(\max(y_0, y_1) + \frac{w_Y}{2}\right)\right)$$

This then determines the set of the candidate tiles and steps 22, 23, and 24 are applied as before.

For the testing of the edges, the present invention only needs to consider the two edges that are parallel to the central axis, as the end caps have effectively been tested as part of the bounding box. These will be called Edge0 and Edge1. The computation of Edge0's a and b values is done as before:

$$a_0 = y_0 - y_1$$
$$M \quad M \quad M$$

$$b_0 = x_1 - x_0$$
$$M \quad M \quad M$$

The computation Edge0's C co-efficient, c_0 now proceeds as follows: The orientation of the edge is determined as described previously, and then the shift amounts are computed using the following method:

```
IF ((Mode == ToTopLeft) OR (Mode == ToBottomLeft))
{
     x_shift = (Tx + w_x/2);
}
ELSE
{
     x_shift = (-w_x /2);
}
IF ((Mode == ToTopLeft) OR (Mode == ToTopRight))
{
     y_shift = (Ty + w_y /2);
}
ELSE
{
     y_shift = (-w_y/2);
}
```

The remainder of the calculation for c_{0M} is then exactly the same as previously, i.e.,…

$$c_{iM} = x_{iM} y_{jM} - x_{jM} y_{iM} + x_{shift}.a_{iM} + y_{shift}.b_{iM} + fslop_i$$

…where $i=0, j=1$.

For edge1, the 'a' and 'b' values are simply negatives of Edge0's values, while its c coefficient is computed using the above procedure.

We now describe the steps taken in unit '16' of the preferred embodiment, i.e. the testing for small primitives that can be rejected during tiling because they will not be sampled by the renderer. Although this will be discussed in reference to triangle primitives, this aspect can also be applied to the line segment primitives but, for brevity, this will not be covered here, as its application will be apparent to those skilled in the art. Similarly, the discussion only

covers 'clockwise' primitives, as the modifications to support the other 'culling' types are also straightforward. The embodiment uses a pixel sampling point that is located at the top left corner of the pixel. Other possibilities, such as choosing the centre of the pixel, could be achieved by offsetting the primitive.

The processing is described with reference to Figure 10. Using the maximum and the minimum values of the primitive, as computed in step '21', the system determines a set of 4 pixel sampling locations, $\{S_{00}, S_{01}, S_{10}, S_{11}\}$, '80', as follows ...

$$s_{x\min} = \lceil p_{x\min} \rceil \quad s_{x\max} = \lfloor p_{x\max} \rfloor$$
$$s_{y\min} = \lceil p_{y\min} \rceil \quad s_{y\max} = \lfloor p_{y\max} \rfloor$$
$$S_{00} = (s_{x\min}, s_{y\min})$$
$$S_{01} = (s_{x\max}, s_{y\min})$$
$$S_{10} = (s_{x\min}, s_{y\max})$$
$$S_{11} = (s_{x\max}, s_{y\max})$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ are the standard 'floor' and 'ceiling' operators. As examples, in Figure 7, S_{00} and S_{10} are indicated by '55' and '56' respectively, while in Figure 9, they are indicated by '73' and '71' respectively. Note that the positions of these appear to swap due to the relative sizes of the bounding boxes and the behavior of 'floor' and 'ceiling'.

In step '81', the bounding box of the primitive is tested to see if it misses all sampling points, i.e. testing for trivial cases, an example of which is shown in Figure 7. If this is the case, the primitive is culled, '82'. The test in '81' can be summarized as:

IF ($s_{x\min} > s_{x\max}$ OR $s_{y\min} > s_{y\max}$) THEN CULL

This test, however, will not cull the examples shown in Figures 8 and 9.

Because it is not cost effective to consider primitives larger than a certain size, the preferred embodiment limits its small object culling tests to those primitives whose

chosen sampling locations are spread out no further than a 1x1 pixel area. Step '83' tests for this condition and if the primitive is deemed to be too large, it is not tested for culling, '84'. The test performed in '83' is:

IF (($s_{x\max} - s_{x\min}$) > 1) OR ($s_{y\max} - s_{y\min}$) > 1) THEN KEEP

The invention now tests each of the four sampling points to see if they are inside the primitive: As before, the edge coefficients, '85' are computed using the previously specified edge equations. In this situation, however, because the range of numbers is so small, the calculations can be done in 'high precision' very cheaply using fixed-point mathematics.

Note that there are at most three possible cases to consider: That of only one distinct sampling point, two distinct sampling points (as shown by '61' and '62' in Figure 8), or four distinct sampling points (as shown by '71', '72', '73', and '74' in Figure 9).

In the preferred embodiment, up to four sample locations can be tested, '86', against each edge quite cheaply by noting that the four test inequalities for an edge for the situation with the four distinct sampling locations are:

$$A_i S_{x\min} + B_i S_{y\min} + C_i \geq 0$$

$$\begin{aligned} & A_i S_{x\max} + B_i S_{y\min} + C_i \geq 0 \\ & \Leftrightarrow A_i (S_{x\min} + 1) + B_i S_{y\min} + C_i \geq 0 \\ & \Leftrightarrow A_i S_{x\min} + B_i S_{y\min} + C_i \geq -A_i \end{aligned}$$

$$\begin{aligned} & A_i S_{x\min} + B_i S_{y\max} + C_i \geq 0 \\ & \Leftrightarrow A_i S_{x\min} + B_i S_{y\min} + C_i \geq -B_i \end{aligned}$$

$$\begin{aligned} & A_i S_{x\max} + B_i S_{y\max} + C_i \geq 0 \\ & \Leftrightarrow A_i S_{x\min} + B_i S_{y\min} + C_i \geq -B_i - A_i \end{aligned}$$

As can be seen, the same basic calculation is reused, i.e. $A_i S_{x\min} + B_i S_{y\min} + C_i$ and then compares it to the four values, $\{0, -A_i, -B_i, (-A_i - B_i)\}$. Since the worst case requires four sampling points and since sampling outside the primitive does no harm, the embodiment, for simplicity, tests all of the above conditions regardless of the number of genuinely interesting sampling points.

A sample point is deemed to be outside the primitive if any of the edges fails its test. In step '87', if all of the sample locations are outside the primitive, then the primitive is culled, in step '82', otherwise it is kept, in step '84'.

In an alternative embodiment, a larger than 2x2 sampling grid could be used.